pya - a Python Library for Audio Processing and Auditory Display

- (c) 2019 by Thomas Hermann and Jiajun Yang
- suplementary material for the paper submitted to Nordic SMC 2019, Sweden
- For full tutorial for pya, please refer to the official source code repository https://github.com/interactive-sonification/pya (https://github.com/interactive-sonification/pya)

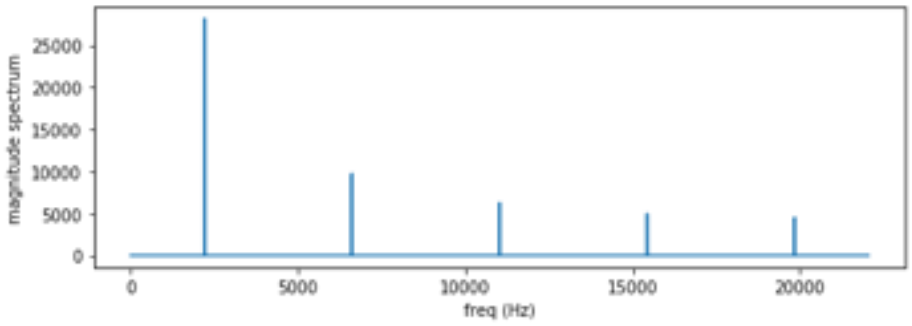
In [6]:
```
# imports
from pya import *
import numpy as np
import matplotlib.pyplot as plt
```

In [7]:
```
s = Aserver(bs=2048)
Aserver.default = s  # set default Aserver to use play() w/o explicit arg
s.boot()
```

Out[7]:
```
AServer: sr: 44100, blocksize: 2048,
         Stream Active: True, Device: Built-in Output, Index: 1
```

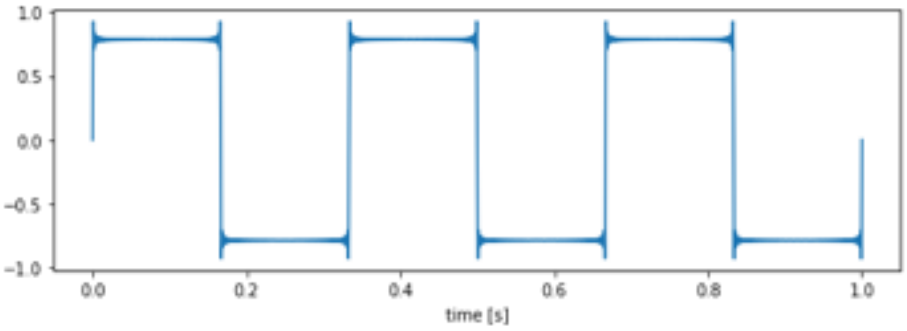
Section 3.3. Aspec

In [8]:
```
plt.figure(figsize=(8,3))
asig = Ugen().square(2205, duty=0.5)
asig.to_spec().plot();
plt.ylabel("magnitude spectrum")
plt.tight_layout()
```



Section 4.1. Sound Synthesis example

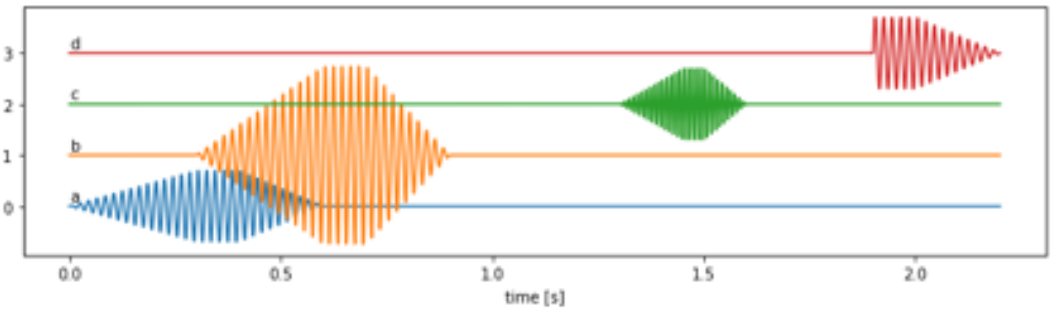
In [5]:
```
plt.figure(figsize=(8,3))
asig = Asig(1.0)
harmonics = (1 + 2 * np.arange(100))
for f, a in zip(3*harmonics, 1/harmonics):
    asig += Ugen().sine(f, a, sr=44100, dur=1.)
asig.label = "Additive Synthesis"
asig.plot(); plt.xlabel('time [s]'); plt.tight_layout()
```



Sec. 4.2. Multichannel Audio

In [9]:
```
a = Asig(1, sr=1000, channels=4, cn=['a', 'b', 'c', 'd'], label='multichannel')
b = Ugen().sine(freq=50, sr=1000, dur=0.6).fade_in(0.3).fade_out(0.2)
a.x[:, 'a'] = 0.2 * b  # no need to extend as len(src)<len(dest)
a.x[300:, 'b'] = 0.5 * b  # extends a to 0.9 seconds
a.x[1300:, 'c'] = 0.2 * b[::2]  # extends a further, writing beyond end
a.x[1900:, 3] = 0.2 * b[300:]  # note that 3 is 'd'
plt.figure(figsize=(12,3)); a.plot(offset=1, scale=3.5)
```

Out[9]:
```
Asig('multichannel'): 4 x 2200 @ 1000Hz = 2.200s cn=['a', 'b', 'c', 'd']
```



Sec 4.3.1 Audification

For this example, we use typical EEG data to be expected for an epileptic seizure,

- 19 channels, sampled at 256.41 Hz,
- data provided as csv file.

We load the data into an Asig as follows:

In [11]:
```
# load data: EEG data
data = np.loadtxt("data/epileptic-eeg.csv", delimiter=",")
chnls = "FP1 FP2 F3 F4 C3 C4 P3 P4 O1 O2 F7 F8 T7 T8 P7 P8 FZ CZ PZ".split(" ")
aeeg = Asig(data, sr=256.41, cn=chnls)
aeeg
```

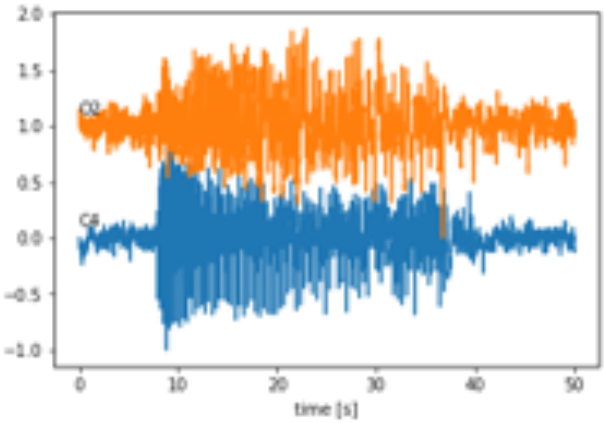
Out[11]:
```
Asig(''): 19 x 12821 @ 256.41Hz = 50.002s cn=['FP1', 'FP2', 'F3', 'F4', 'C3',
'C4', 'P3', 'P4', 'O1', 'O2', 'F7', 'F8', 'T7', 'T8', 'P7', 'P8', 'FZ', 'CZ',
'PZ']
```

Note that Asigs are also very suitable to represent such multi-dimensional time series.

Now let's listen (and look) to an audification of channel 5 and 9 at a 10x-speedup:

In [12]:
```
a1 = aeeg[:,[5, 9]].norm(1).plot(offset=1).resample(44100, rate=10).play()
a1.save_wavfile('media/S1-eeg-absence-rate10.wav')
```

Out[12]:
```
Asig('_arrayindexed_normalised_resampled'): 2 x 220861 @ 44100Hz = 5.008s c
n=['C4', 'O2']
```



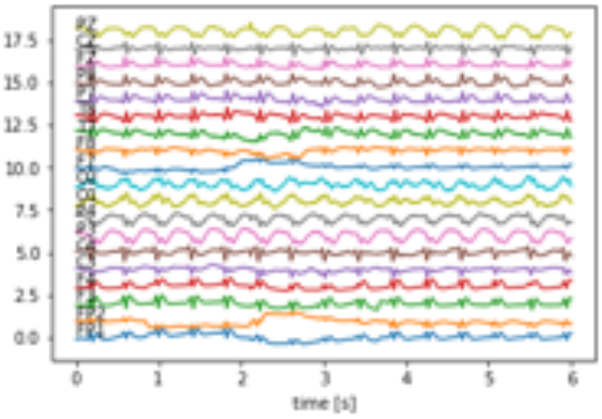
We hear the rhythmical (3/sec) epileptic pattern arising and falling back into background signal. Listening at higher speedup (25x), we discover the ritardando:

In [13]:
```
a1 = aeeg[:,[5, 9]].norm(1).resample(44100, rate=25).play()
# pitch decrease corresponds to slowing down of epileptic rhythm over seizure
a1.save_wavfile('media/S2-eeg-absence-rate25.wav')
```

Out[13]:
```
Asig('_arrayindexed_normalised_resampled'): 2 x 88344 @ 44100Hz = 2.003s cn=['
C4', 'O2']
```

Now let's select 6 seconds (from 16s - 22s), to listen next in 'slow motion'. We resample to a modest 8000 Hz audio rate

```
In [14]: a2 = aeeg[{16:22}].norm().plot(scale=0.5, offset=1).resample(8000, rate=1)
```

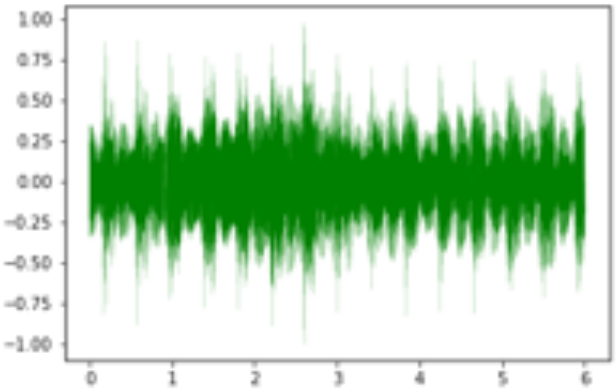


However, we need to modulate the signal with a sine since otherwise we could not perceive the low frequencies.

Let's

- use integer multiples of 90 Hz for the series of channels,
- and listen to the result in 0.5x the original speed
- and plot the compiled audio in green with a thin line
- and save the audio file to a wav file.

```
In [15]: asum = Asig(a2.get_duration(), sr=a2.sr)
         for i in range(a2.channels):
             asum += a2[:, i] * Ugen().sine(90*(i+1), dur=a2.get_duration(), sr=a2.sr)
         asum.norm().plot(color="g", lw=0.1).stereo().play(rate=0.5).save_wavfile('media
         /S3-timbraleeg.wav')
         asum.label='timbral-eeg-son'; asum
Out[15]: Asig('timbral-eeg-son'): 1 x 48016 @ 8000Hz = 6.002s cn=['0']
```



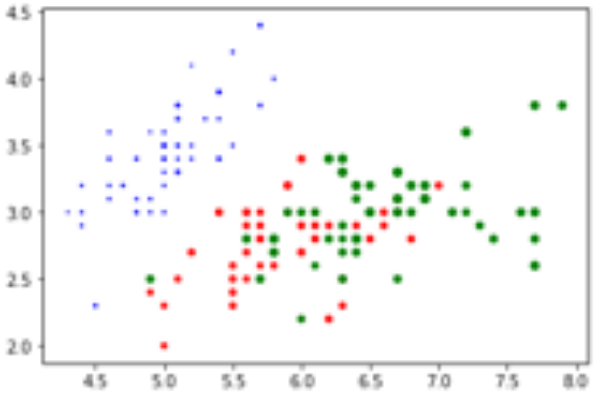
Obviously, quite complex tasks can be easily expressed in short and concise code.

Section 4.3.2. Parameter Mapping Sonification

We demonstrate pya for discrete parameter mapping sonification, i.e. data variables are mapped to synthesis parameters and resulting sounds added to the sound track. Our parameters are

- onset,
- frequency,
- duration,
- amplitude
- panning For the demo we use the famous Iris data set which features 4 geometric measurements of iris plants and their class label as 5th feature with values 0..2. Let's first load the data set

In [16]:
```
diris = np.loadtxt("data/iris.csv", delimiter=",")
plt.scatter(diris[:,0], diris[:,1], c=[['b','r','g'][int(el)] for el in diris
[:,4]], s=10*diris[:,3]);
```

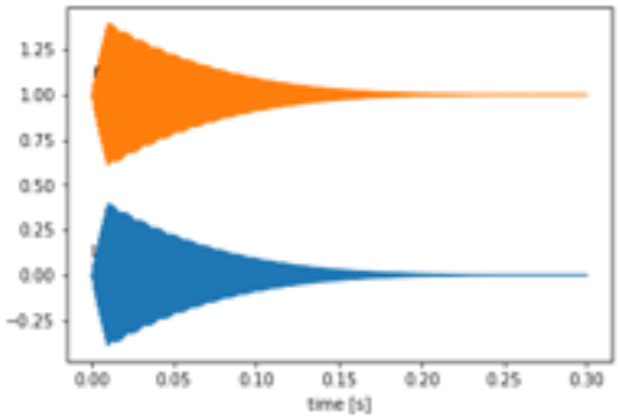


Next let's test our synth, a sine pulse with controlled panning

In [18]:
```
def mysyn(freq=440, dur=0.34, amp=0.9, att=0.01, pan=0.5):
    return Ugen().sine(freq, dur=float(dur), sr=8000).fade_in(0.01)\
.envelope([0,1,0], [0, att, dur], curve=4).gain(amp).stereo([1-pan, pan])

# test synth: plot and play a test tone
mysyn(freq=900, dur=0.3, att=0.001).plot(offset=1).play()
```

Out[18]:
```
Asig('sine_fadein_enveloped_scaled_to_stereo'): 2 x 2400 @ 8000Hz = 0.300s c
n=['l', 'r']
```



Finally we apply the mapping by iterating over the data

- computing parameters by a linear mapping from the features
- creating the sound and adding it to the sonification track son
- finally normalizing, plotting and plying the result

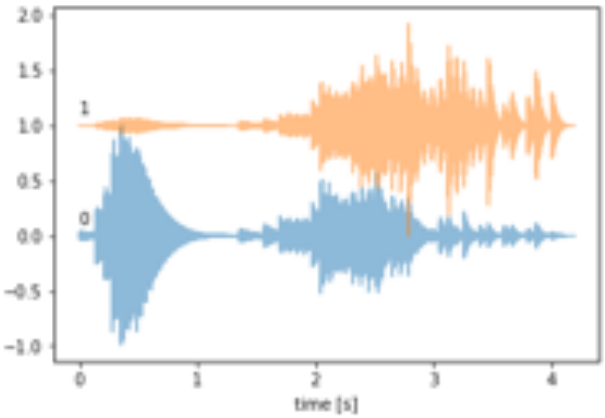
Note that the sound computing is extremely condensed:

- complex operations happen in only 2-3 lines of code

In [19]:
```
dmin = np.min(diris, axis=0)
dmax = np.max(diris, axis=0)
def mapcol(vec, k, rmin, rmax):
    return linlin(vec[k], dmin[k], dmax[k], rmin, rmax)
```

In [20]:
```
son = Asig(1.0, sr=8000, channels=2)
for i, v in enumerate(diris):
    onset = mapcol(v, 2, 0,4)
    dur = mapcol(v, 4, 1, 0.2)
    freq = mapcol(v, 1, 200, 2000)
    amp = mapcol(v, 0, 0, 1)
    pan = mapcol(v, 3, 0, 1)
    son.x[{onset:None}] += mysyn(freq, dur, amp, 0.001, pan)
son = son.norm().plot(offset=1, alpha=0.5).play()
son.save_wavfile('media/S4-PMson-iris-dataset.wav')
```

Out[20]:
```
Asig('_normalised'): 2 x 33599 @ 8000Hz = 4.200s cn=['0', '1']
```



In [22]:
```
# create Figure for paper
plt.figure(figsize=(8,2.5))
son[:,1].to_stft().plot(ampdb)
plt.xlabel('time [s]'); plt.ylabel('frequency (Hz)');
```